Finding Near-Min Cuts

Kevin Wood kwood@nps.navy.mil
Operations Research Department
Naval Postgraduate School

Oct 2002

Purpose of this talk

- **Describe an algorithm to enumerate all *s*-*t* cuts in a directed network whose capacity is within 1+ε of being optimal for $\varepsilon \geq 0$. ("near-min cuts")**
- **Prove that the run time is polynomial per cut enumerated when $\varepsilon = 0$.**
- **Prove that it is polynomial for certain graph topologies .**

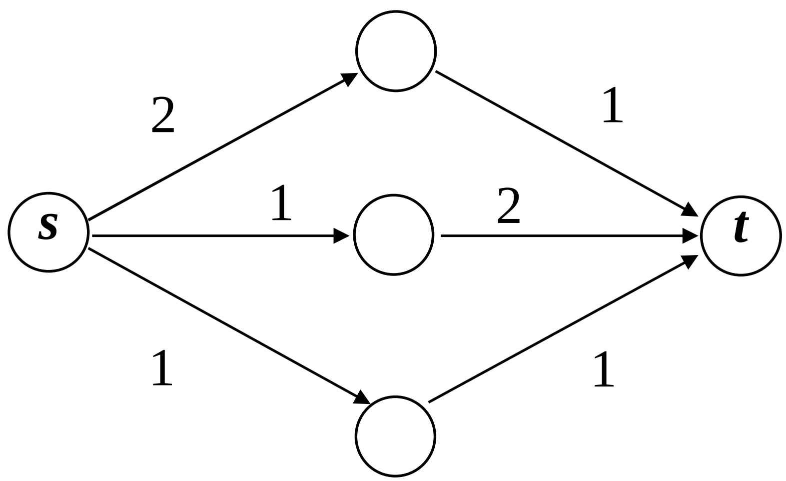
A Network Interdiction Problem

- **Given network G=(*N*,*A*) and resource r_k required to "destroy" *k*=(*i,j*), find the minimum total resource required to cut all comm between nodes *s* and *t* .**
- **Simple solution via the max-flow min-cut theorem:**

  Set resources as arc capacities, find max flow and min capacity cut.

Max-flow, min-cut

(With multiple solutions)



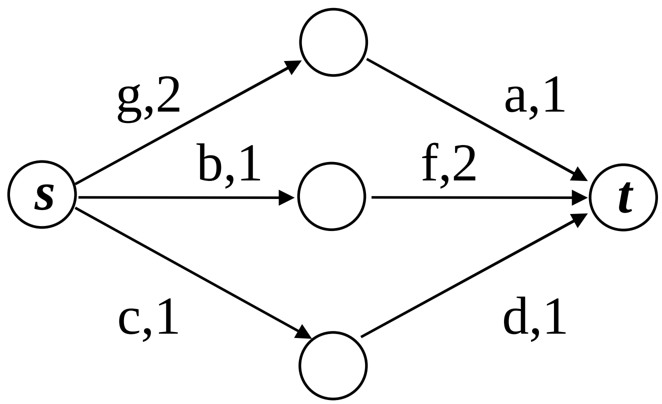
One min cut easy to find, but…

- **There may be secondary considerations, e.g., collateral damage, logical constraints**
- **So, find <u>all</u> <u>min</u> <u>cuts</u> and evaluate against other relevant criteria**
- **How to enumerate?**
  - Brute force: Enumerate <u>all</u> <u>cuts</u>
  - Some theoretical work in literature
  - Practical: Norm Curet, Applied Math, NSA

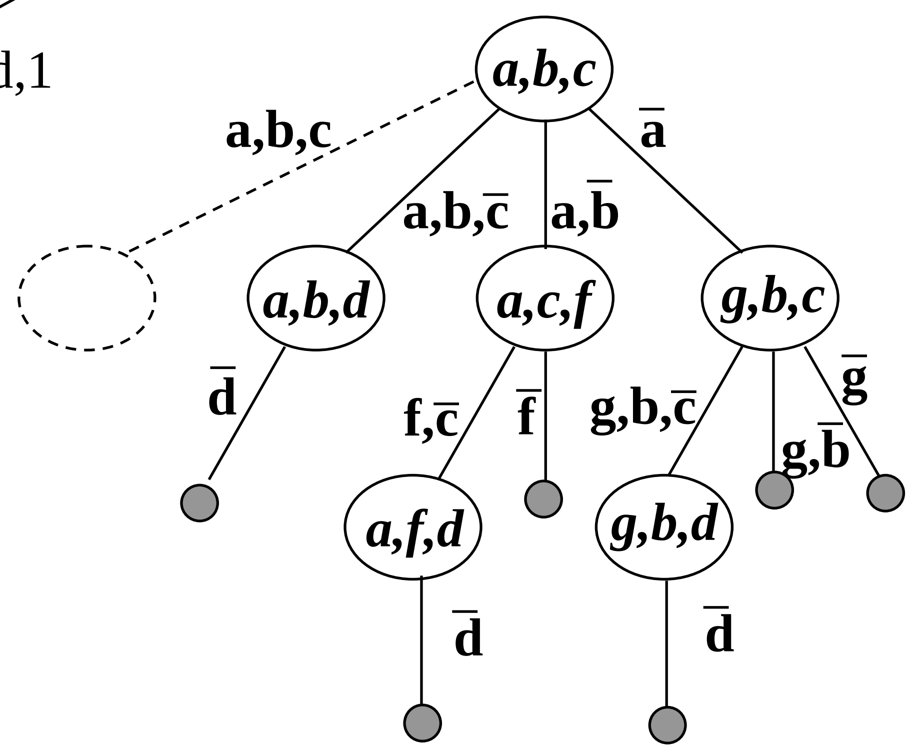
The next refinement

- Allow near-optimal solutions, i.e., accept near-min cuts.
- Can still enumerate all cuts!
- Some graph theoretical work. Ramanathan and Colbourn (1987), Vazirani and Yannakakis (1997) enumerate *cutsets*
- Two Masters theses at NPS.

Our partitioning scheme



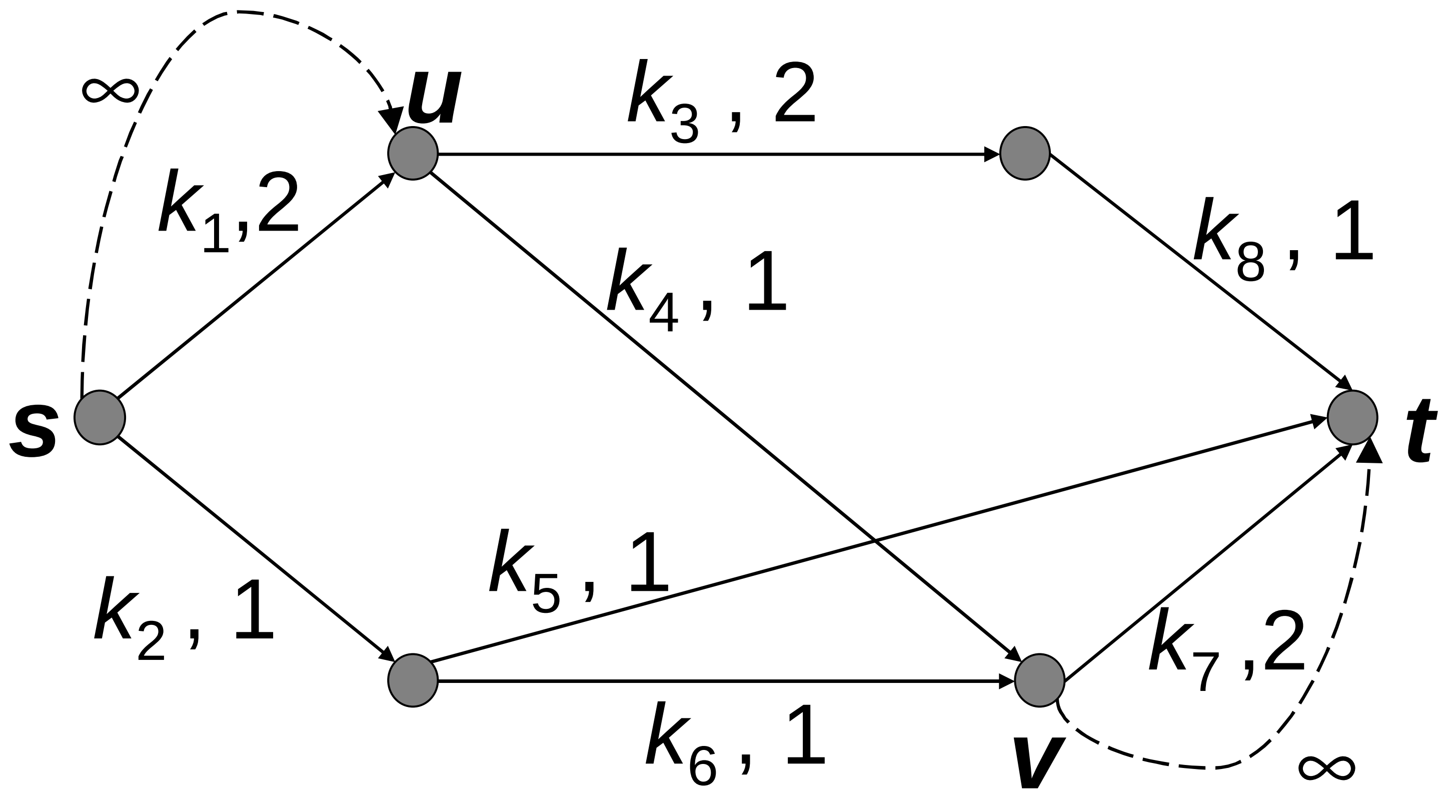
Backtrack if +1 tolerance exceeded



Forcing arcs in and out of cuts

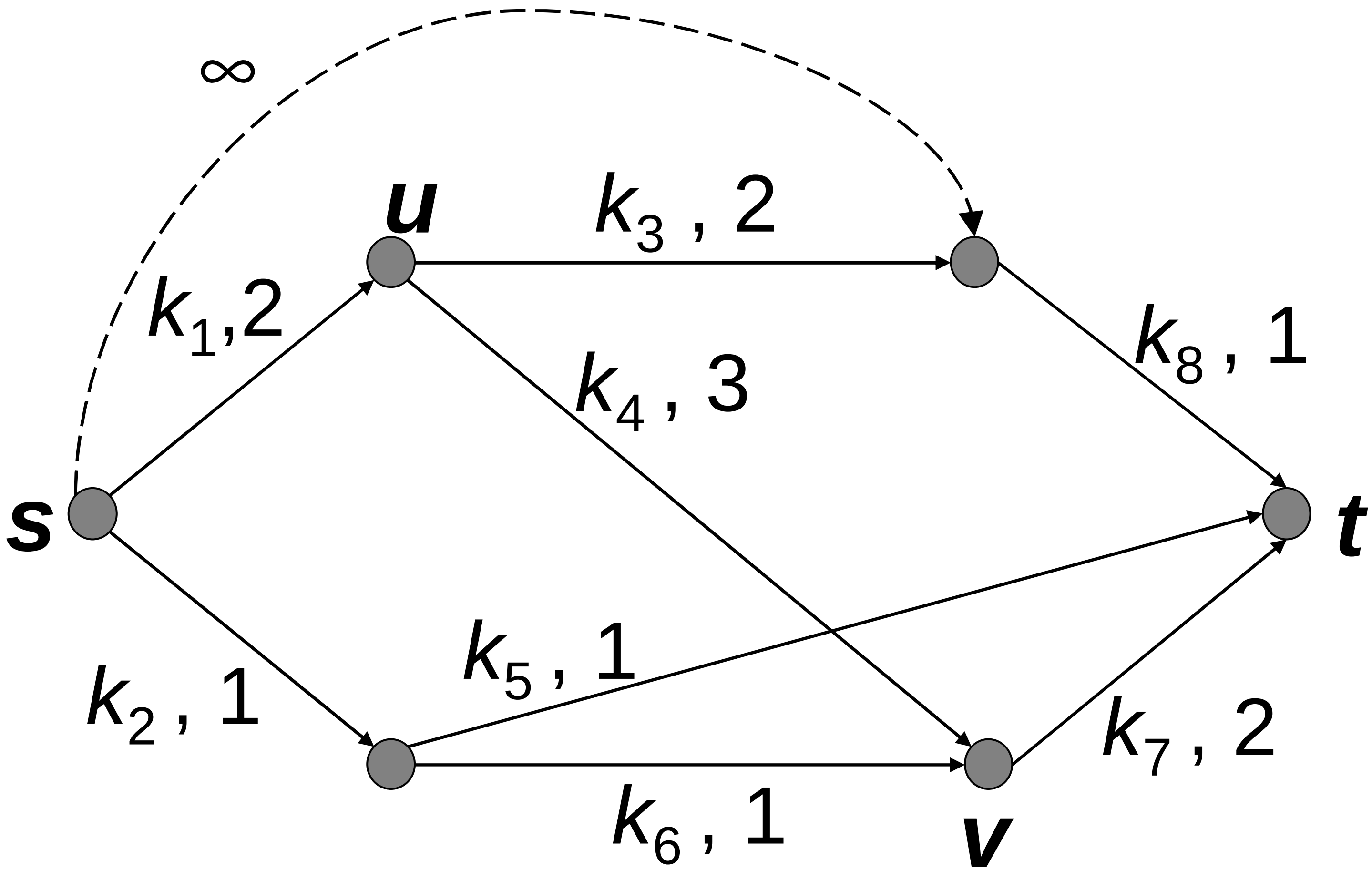
- To ensure that a given arc k is excluded from all cuts below a given tree node, set $u_k = \infty$
- To ensure inclusion of $k = (i,j)$, add (s,i) and (j,t) with capacities of ∞ (treat i as an extra source and j as an extra sink)
- Exclusion always works; inclusion can introduce new "pseudo-minimal" cuts
- Just keep track of A_{IN} and A_{OUT}

Force inclusion of arc $k_4 = (u,v)$



Actually, just treat *u* as a new source and *v* as a new sink.

Creating pseudo-minimal cuts ☹



{k_1, k_2, k_8} is a pseudo-minimal cut

An Algorithm

MAIN

Input: $G = (N, A)$, $\underline{u}$, ε, s, t */* Global */*

Output: All cuts with cap. $\leq (1 + \varepsilon)\, z_{min}$

{

$(z_{min}, A_C) \leftarrow$ **Maxflow** $(G, \{s\}, \{t\}, \underline{u})$;

/* z_{min} **is also global */**

$A_{IN} \leftarrow \emptyset$; $A_{OUT} \leftarrow \emptyset$;

Enumerate(A_{IN}, A_{OUT});

}

Algorithm: recursive routine

Enumerate(A_{IN}, A_{OUT})

{ $\underline{u}' \leftarrow \underline{u}$; $u_k \leftarrow \infty\ \forall k \in A_{OUT}$;

$S \leftarrow \{s\}+\{i \mid (i,j) \in A_{IN}\}$; $T \leftarrow \{t\}+\{j \mid (i,j) \in A_{IN}\}$;

(z', A_C) $\leftarrow$ **Maxflow** (G, S, T, $\underline{u}'$);

If ($z' > (1 + \varepsilon) \times z_{min}$) **return**;

If(A_C is minimal) **Print** (z', A_C);

For (each $k \in A_C - A_{IN}$){

$A_{OUT} \leftarrow A_{OUT}+\{k\}$;

Enumerate(A_{IN}, A_{OUT});

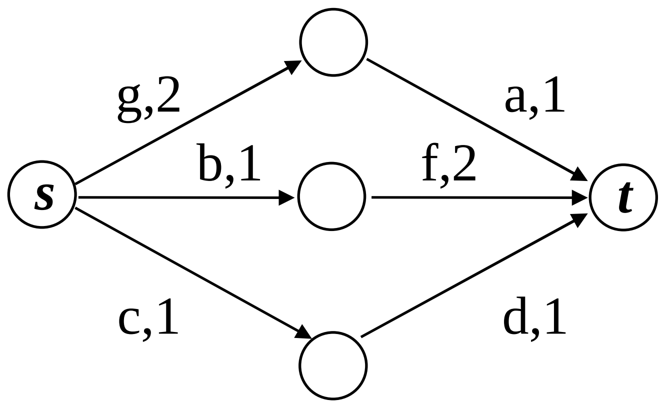
$A_{OUT} \leftarrow A_{OUT} - \{k\}$; $A_{IN} \leftarrow A_{IN} + \{k\}$;

}

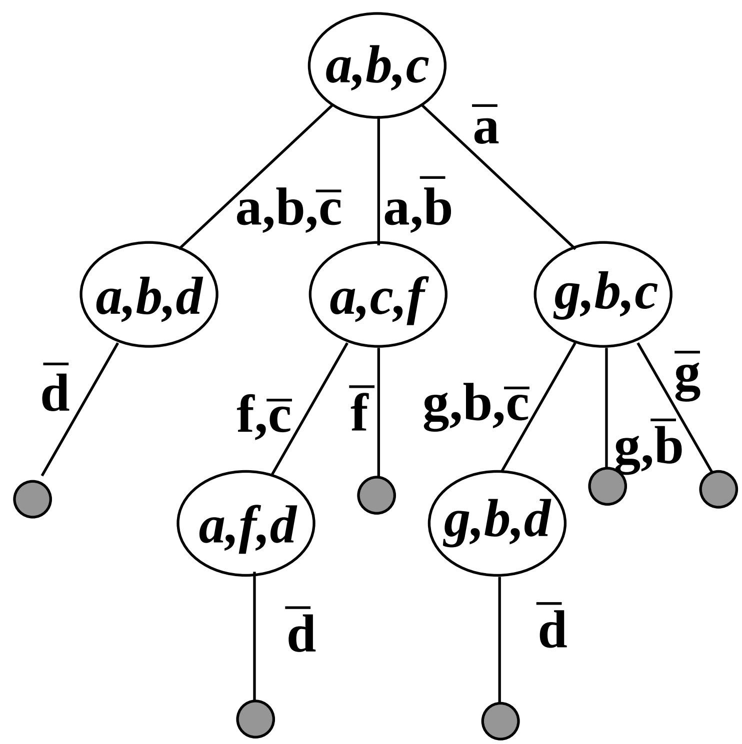
return;

}

An Example



Backtrack if +1 tolerance exceeded



Details on a chalkboard!

Algorithm: Notes

- Actual implementation more efficient
- Max-flows not solved from scratch; use a flow-augmenting path algorithm
- Pre-emptive backtracking from within Maxflow allowed
- Work per iteration O(|A|) for finding min cuts; “usually” O(|A|) anyway?
- Testing for non-minimal cuts O(|A|)

Algorithm: Thm 1 (it works!)

- **Theorem 1: The algorithm enumerates all near-min cuts.**

  Proof: Simple partitioning and enumeration argument. ***QED***

Let *C* denote the set of near-min cuts and let MF denote time for max flow

Note: The enumeration tree has only "productive tree nodes" (near-min cuts) or "unproductive tree nodes"

Algorithm: Thm 2 (efficiency)

- **Theorem 2: Run time is $O(|A||N||C|+\mathrm{MF})$ for finding min cuts A_C (with pre-emptive backtracking).**

  Proof: Initial cut found in $O(\mathrm{MF})$ time. A new min cut is generated & proven min in two $O(|A|)$ flow augmentations, or pre-emptive backtracking occurs. There are at most $|N|$ dead tree nodes for each productive node and |Productive nodes|=$|C|$. *QED*

Algorithm: Thm 3 (efficiency)

- **Theorem 3: ($\varepsilon > 0$) Run time is $O(|A||N||C|+\mathrm{MF})$ for finding near-min cuts A_C when $z_{min}\varepsilon < u_{min}$.**

  Proof: Same as previous proof, essentially, because any cut with capacity $z_{min}(1+\varepsilon)$ must be minimal:

  The smallest capacity a non-min'l cut can have is $z_{min}+u_{min} > z_{min}(1+\varepsilon)$, so any non-min'l cut causes a backtrack. *QED*

Algorithm: Thm 4 (efficiency)

- **Theorem 3: Run time is *O*(|*N*||*C*| MF) for finding near-min cuts if all arcs of the form (s,v) and (v,t) exist.**

  Proof: Quasi-inclusion does not change the connectivity of *G* under these conditions. Every cut found is minimal. Each productive node in the enumeration tree has at most |N|-1 nonproductive children. *QED*

- **Corollary 1: Run time is *O*(|*N*||*C*| MF) for finding near-min cuts in complete graphs.**

Algorithm: General Efficiency

- My guess: Difficult
- Problematic examples exist. Non-minimal cuts can be produced when forcing arcs in
- Finding a minimal cut that includes certain arcs and excludes others is an NP-complete problem
- There is room for improved efficiency by identifying “non-forceinable arcs”!

Enhancements and results

- **Don't solve max flows from scratch**
- **Results: 733 MHz Pentium III in Java**
- **Only results for grid networks here, with $c_k = 1$**
- **All 249 min cuts in a 25 by 250 grid ($|N|$=6,252, $|A|$=24,500) in 18 seconds**
- **All 431,728 near-min cuts ($\varepsilon = 0.15$) in a 25 by 25 grid in 973 seconds (253 non-minimal cuts encountered)**

Further research

- Find more classes of graphs that admit efficient solutions
- Add tests for “edge domination” to eliminate certain edges from possible quasi-inclusion
- Using the basic algorithm in a “network diversion problem”: Find a min-weight, minimal cut that contains a given edge